

SELF JOIN ELIMINATION THROUGH UNION

BACKGROUND OF THE INVENTION

1. Field of the Invention.

5 This invention relates in general to database management systems performed by computers, and in particular, to the optimization of queries that include self joins.

2. Description of Related Art.

10 Since its introduction, XML, the eXtended Markup Language, has quickly emerged as a universal format for publishing and exchanging data over the World Wide Web. However, problems still exist in publishing data from object-relational databases as XML documents.

15 In the business-to-business e-commerce area, there is a widely recognized need to create XML documents by combining one or more object-relational tables, e.g., by creating an XML purchase order by joining a customer with information drawn from other tables. A relational join is a well known operation that combines information from two base tables by creating pairs of matching rows that are related using a predicate.

20 In comparison with simple select queries over a single table, join queries are costly in terms of system performance and much research has been done to optimize these queries. Query rewrite optimizations can sometimes be used to transform join queries into simple select queries.

For example, if a join is a self join, and the join predicate links the two quantifiers on the table's key columns, the query can be rewritten into a simple select. This is illustrated using the following example:

25

```
SELECT E1.SAL, E2.SAL
FROM EMP E1, EMP E2
WHERE E1.NO = E2.NO
```

09987759 "052101
T07290" 6578860

The query selects the salaries of employees on matching values of attribute NO. Since NO represents employee numbers and it is also the table's key, each row represented by quantifier E1 will only match with itself in E2. Therefore, the above
5 query can safely be rewritten as the following query, preserving its semantics:

```
SELECT E1.SAL, E1.SAL  
FROM EMP E1
```

10 The principle of self join elimination can be extended beyond simple queries, such as the example above, to include more complex queries. Commercial database management systems, such as DataBase 2 (DB2™) Universal DataBase (UDB™) sold by IBM Corporation, the assignee of the present invention, implement a number of different query rewrite transformations, which enhances the performance of such
15 queries.

However, there is still a need for improved techniques at optimizing self joins. Specifically, there is a need in the art for transformations of self joins that are transitively derived through table expressions which themselves cannot be simplified using a SELECT-MERGE query rewrite optimization.

20

SUMMARY OF THE INVENTION

To overcome the limitations in the prior art described above, and to overcome other limitations that will become apparent upon reading and understanding the present specification, the present invention discloses a method, apparatus, and article of
25 manufacture for optimizing self join queries. The query optimization technique determines whether a query includes a self join that is transitively derived through table expressions having UNION operators. If so, the query is simplified to eliminate the table expressions and to reduce the query to an equivalent query over the tables.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

5 FIG. 1 illustrates the computer hardware environment of the present invention;

FIG. 2 is a flowchart illustrating the steps necessary for the optimization and execution of SQL statements according to the preferred embodiment of the present invention;

10 FIG. 3 is a block diagram that illustrates the data structure used to maintain a derivation list of quantified columns according to the preferred embodiment of the present invention;

FIG. 4 is a block diagram that illustrates the data structure used to maintain a derivation list of Subset columns according to the preferred embodiment of the present invention; and

15 FIGS. 5A-5D together are a flowchart illustrating the method of optimizing queries in step 202 of FIG. 2 according to the preferred embodiment of the present invention

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

20 In the following description of the preferred embodiment, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration a specific embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural and functional changes may be made without departing from the scope of the present invention.

25

OVERVIEW

An XML middleware system presents relational databases as XML repositories. Queries against the XML representation of the databases are expressed in X-Query, a

query language developed to query XML documents. The XML middleware system may optimize the X-Query statements using an optimization technique known as Self Join Elimination Through Union. The XML middleware system translates the optimized X-Query statements into SQL queries that can be executed by a relational database management system (RDBMS). When the query results are returned by the RDBMS, the XML middleware system renders the query results as one or more XML documents.

HARDWARE AND SOFTWARE ENVIRONMENT

FIG. 1 illustrates an exemplary hardware and software environment that could be used with the preferred embodiment of the present invention. In the exemplary environment, a computer 100 executes an XML (eXtended Markup Language) middleware system 102 that interfaces to an object-relational database management system (RDBMS) 104 that manages one or more databases.

In the preferred embodiment of the present invention, the XML middleware system 102 and the RDBMS 104 may be executed on the same computer 100 or different computers 100. Moreover, the computer 100 may comprise a server on a network, such as the Internet, wherein the computer 100 is accessed by one or more client computers connected to the network. Those skilled in the art will recognize that the present invention can be used in any number of different hardware and software environments.

Also, in the preferred embodiment of the present invention, the XML middleware system 102 comprises the XPERANTO (Xml Publishing of Entities, Relationships, ANd Typed Objects) system offered by IBM Corporation, and the RDBMS 104 comprises the DataBase 2 (DB2™) Universal DataBase (UDB™) product offered by IBM Corporation. Those skilled in the art will recognize that the present invention has application to any XML middleware system and any DBMS, and may be implemented in many different types and brands of such systems.

Generally, the XML middleware system 102 and the RDBMS 104 each

comprises logic and/or data that is embodied in or retrievable from a device, medium, signal or carrier, e.g., a fixed or removable data storage device, a remote device coupled to the computer by a data communications device, etc. Moreover, this logic and/or data, when read, executed, and/or interpreted by the computer 100, cause the computer 100 to perform the steps necessary to implement and/or use the present invention.

Thus, the present invention may be implemented as a method, apparatus, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" as used herein is intended to encompass logic and/or data embodied in or accessible from any device, medium, signal or carrier.

Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention. For example, those skilled in the art will recognize that any combination of the above components, or any number of different components, including computer programs, peripherals, and other devices, may be used to implement the present invention, so long as similar functions are performed thereby.

OPERATION OF THE XML MIDDLEWARE SYSTEM AND RDBMS

The XML middleware system 102 provides a default XML view of existing relational databases managed by the RDBMS 104 and an XML query facility with which users can define new XML views. These views can also be queried using the same XML query facility, without the users having to learn or write SQL. The XML middleware system 102 translates XML queries into SQL queries, submits the SQL queries to the RDBMS 104 for execution, receives the results from the RDBMS 104, and then translates these results into XML documents.

As shown in FIG. 1, the XML middleware system 102 is organized into four components: Query Translation 106, XML View Services 108, XML Metadata Module 110, and XML Tagger 112. The Query Translation component 106 is organized into

three sub-components: X-Query Parser 114, Query Rewrite 116, and SQL Translation 118. The functions performed by each of the components and sub-components are described below.

- 5 • Query Translation 106: Translates the XML query language (X-Query) into an appropriate dialect of SQL for the RDBMS 104. This function is further sub-divided among the sub-components of Query Translation 106:
 - 10 ▪ X-Query Parser 114: Takes X-Query statements and generates an XQGM (XML Query Graph Model), which is a language-neutral intermediate representation for the X-Query statements.
 - 15 ▪ Query Rewrite 116: Optimizes the XQGM, resolves view references, performs XML view composition, and produces a semantically equivalent representation of the XQGM.
 - 20 ▪ SQL Translation 118: Translates the XQGM to SQL statements for use by the RDBMS 104.
- XML View Services 108: Serves as a storage and retrieval interface for X-Query view definitions. When views are defined, they are stored in a dedicated table. They can be later retrieved for view unfolding.
- XML Metadata Module 110: Takes catalog information and produces XML schema information 120 for XML views and query results.
- XML Tagger 112: Converts SQL query results into XML results 122, which typically comprise structured XML documents.

As shown in FIG. 1, the RDBMS 104 is organized into three components: SQL Query Processor 124, Stored Tables 126, and System Catalog 128. The functions performed by each of the components is described below.

- SQL Query Processor 124: Processes and executes the SQL query received from Query Translation 106 and provides the results from that execution to the XML Tagger 112.

- Stored Tables 126: The data stored in the relational database.
- System Catalog 128: Meta-data concerning the Stored Tables 126.

QUERY OPTIMIZATION AND EXECUTION

5 FIG. 2 is a flowchart illustrating the steps necessary for the optimization and execution of X-Query statements according to the preferred embodiment of the present invention. Block 200 represents the input of X-Query statements into the XML middleware system 102. Block 202 represents the XML middleware system 102 interpreting the X-Query statements. Block 204 represents the XML middleware system 102 generating the XQGM runtime structures from the X-Query statements.

10 An optimization function at Block 204 may transform or optimize the X-Query statements in a manner described in more detail later in this specification. Generally, the X-Query statements received as input specify only the desired data, but not how to retrieve the data. This step considers both the available access paths (indexes, 15 sequential reads, etc.) and system held statistics on the data to be accessed (the size of the table, the number of distinct values in a particular column, etc.), to choose what it considers to be the most efficient access path for the query.

20 Block 206 represents the XML middleware system 102 translating the XQGM into SQL statements and providing those SQL statements to the RDBMS 104 for execution. Block 208 represents the RDBMS 104 executing the SQL statements and providing the results therefrom back to the XML middleware system 102. Block 210 represents the XML middleware system 102 translating the results received from the RDBMS 104 into XML documents.

25 DESCRIPTION OF THE OPTIMIZATION TECHNIQUE

 The present invention discloses an improved optimization technique that is typically performed at step 204 of FIG. 2 by the XML middleware system 102. As noted above, the improved optimization technique is known as Self Join Elimination

Through Union.

The following query illustrates the self join elimination transformation through table expressions having UNION operators:

5 SELECT E1.SAL, E2.SAL
 FROM (SELECT E1.NO, E1.SAL
 FROM EMP E3
 UNION
 SELECT E4.NO, E4.SAL
10 FROM RICHEMP E4) AS E1 (NO, SAL),
 (SELECT E5.NO, E5.SAL
 FROM EMP E5
 UNION
 SELECT E6.NO, E6.SAL
15 FROM RICHEMP E6) AS E2 (NO, SAL)
 WHERE E1.NO = E2.NO AND E1.SAL = E2.SAL

The above query can be safely transformed into the following query, preserving its semantics:

20 SELECT E1.SAL, E1.SAL
 FROM (SELECT E3.NO, E3.SAL
 FROM EMP E3
 UNION
25 SELECT E4.NO, E4.SAL
 FROM RICHEMP E4) AS E1 (NO, SAL)

Since UNION is a duplicate eliminating operation, all result rows of the UNION

have a unique combination of values across the list of attributes that are the UNION's result. In other words, a key can be inferred from the UNION, wherein the key contains all attributes in the UNION.

Other combinations of operators can produce these properties. For example, a
5 UNION might not be a duplicate eliminating UNION (a UNION ALL operation does not eliminate duplicates), but can be a table expression in another duplicate eliminating operator.

For example, the following query illustrates these properties:

10 SELECT E1.SAL, E2.SA1
FROM (SELECT E.NO, E.SAL
FROM (SELECT E.NO, E.SAL
FROM EMP E
UNION ALL
15 SELECT E.NO, E.SAL
FROM RICEMP E) AS E (NO, SAL)
GROUP BY E.NO, E.SAL) AS E1 (NO, SAL),
(SELECT E.NO, E.SAL
FROM (SELECT E.NO, E.SAL
20 FROM EMP E
UNION ALL
SELECT E.NO, E.SAL
FROM RICEMP E) AS A (NO, SAL)
GROUP BY E.NO, E.SAL) AS E2 (NO, SAL)
25 WHERE E1.NO = E2.NO AND E1.SAL = E2.SAL

The GROUP BY clause in the table expression guarantees that the combination of values produced by the UNION operation are distinct. The above query can be safely

transformed into the following query:

```
SELECT E1.SAL, E1.SAL
FROM (SELECT E.NO, E.SAL
5      FROM (SELECT E.NO, E.SAL
              FROM EMP E
              UNION ALL
              SELECT E.NO, E.SAL
              FROM RICEMP E) AS E (NO, SAL)
10     GROUP BY E.NO, E.SAL) AS E1 (NO, SAL),
```

While it is unlikely that query authors will compose such equi-join queries, this class of queries can result from internal transformations over views. This is the case for the XML middleware system 102 as described in the preferred embodiment.

15 In the preferred embodiment, self-join elimination through UNION is implemented as a rule-based transformation in the XML middleware system 102. As such, the transformation has two parts. The first part is the condition of the rule, and the second part is the action of the rule. Most rule-based systems are organized similarly.

The rule is performed for each quantifier (i.e., each quantifier in the query graph is individually passed as a parameter to the rule's condition and action). For each 20 quantifier, the rule determines whether the quantifier is part of a self-join that can be eliminated.

The condition of the rule determines whether sufficient conditions for self-join elimination have been detected. These conditions include testing for the presence of self-join through UNION, but can also accommodate simpler queries. 25

The action part of the rule eliminates the quantifier from the query. The transformation includes rewriting expressions such that the quantified columns to be eliminated are substituted by other quantified columns in the same equivalence class.

Once all quantified columns have been rewritten, the quantifier can be removed from the query.

Equivalence among quantified columns is determined by the presence of join predicates establishing equality among quantified columns. These predicate

- 5 transformations also ensure that the query preserves its semantics with respect to null valued columns. For example, if the predicate "E1.NO = E2.NO" is transformed into the predicate "E1.NO = E1.NO", bindings for E1.NO which are null are excluded from the result in both cases.

- Before presenting the general algorithm for self join elimination through
10 UNION, the data structure used to maintain the derivation list of quantified columns is described. FIG. 3 is a block diagram that illustrates the data structure used to maintain a derivation list 300 of quantified columns according to the preferred embodiment of the present invention. Each element 302A, B, and C in the derivation list 300 is itself a list 304A, B, and C, which represents alternatives introduced due to UNION operators.
15 Specifically, each list 304A, B, and C is a list of quantified columns. There is a single quantifier represented per list 304A, B, and C.

- If no UNION operators are present, then each element 302A, B, C of the derivation list 300 contains a single list 304A, B, C of quantified columns (the first list 304A in the figure is an example). The next element 302A, B, C in the derivation list
20 300 is computed by examining the view or table expression over which a quantifier Q is derived, and for quantified columns in a list, the lower columns C from which the columns are derived are computed.

- The list 304A, B, C is only considered for further processing if each lower column is implemented as either a column or a constant (i.e., columns implemented as
25 complex expressions exclude the list of columns from the derivation list 300). For each column in C, the list of equivalence classes is computed and the equivalence classes are divided among quantifiers.

A quantifier is only considered for further processing if it is represented in each

equivalence class. For each remaining quantifier, the list of columns in each equivalence class is computed and these are added as the next element in the list of derivations.

5 If a UNION operator is present, then, for each quantifier over a UNION, the following element in the list of derivations has a number of lists equal to the number of branches in the UNION (i.e., each is obtained by following a branch of the UNION).

Two types of lists of derivations are computed using this method: Superset columns and Subset columns. The derivation list obtained for Superset columns does not allow predicates to appear between each level of the derivation, i.e., if a set of
10 columns is derived from a view having a predicate, that branch of the derivation is not further explored. However, this restriction can be relaxed for predicates which are known not to restrict results belonging to a quantifier, e.g., a key-foreign key join in which the foreign key excludes null values.

The derivation list for Subset columns allows predicates to restrict the result
15 between any level of the derivation. Aside from predicates, outer joins can also affect the derivation lists such that a quantifier bound to the left side of a left outer join operation does not restrict the bindings for that quantifier.

FIG. 4 is a block diagram that illustrates the data structure used to maintain a derivation list 400 of Subset columns according to the preferred embodiment of the
20 present invention. The first element 402A of the list 400 contains a list 404A comprising a column pair "E1.NO, E1.SAL," which is a key. The following element 402B contains two lists in 404B since it involves a UNION. The first list "E3.NO, E3.SAL" is obtained by following one branch of the UNION and the second list "E4.NO, E4.SAL" is derived by following the second branch of the UNION. The
25 horizontal lines between the lists 404A, 404B, and 404C show the derivation paths for the column lists 404A, B, and C. The last element 402C in the derivation list 400 contains a list 404C of the columns belonging to the base tables EMP and RICHEMP that are referenced in respective branches of the UNION operation. Note that the

derivation list for Superset columns is organized similarly to the example of FIG. 4 for Subset columns and is therefore not shown.

The general algorithm for this rewrite is described in terms of this data structure and it is separated into a condition part and an action part:

- 5 For each quantifier Q_i in a query Q , if the condition part is true, then the action part is performed:

Condition

1. Q_i is a FOR EACH quantifier. This is in contrast to, for example, an
10 existential quantifier. In the example, quantifiers E_1 and E_2 qualify this test. In the
 sequel, it is assumed that quantifier E_2 is being tested under the following conditions.
2. Q_i is defined within a SELECT operator that implements an inner join.
 In the example, both quantifiers E_1 and E_2 are in a SELECT box with an inner join.
3. There are at least two quantifiers in the SELECT operator. In the
15 example, the SELECT containing E_2 has two quantifiers.
4. Equivalence classes among columns in the SELECT operator are divided
 into three classes:
- A. Type 1 equivalence classes are sets of quantified columns over
 quantifiers other than Q_i . There are no Type 1 equivalence classes in the
20 example.
- B. Type 2 equivalence classes are sets of quantified columns that
 reference Q_i and at least one other quantifier Q_j . There are two Type 2
 equivalence classes in the example:
 EQV1: { $E_1.NO$, $E_2.NO$ }, EQV2: { $E_1.SAL$, $E_2.SAL$ }.
- 25 C. Type 3 equivalence classes are sets of quantified columns that
 only reference Q_i . If there are Type 3 equivalence classes, then the rule fails.
 There are no Type 3 equivalence classes in the example.
5. For each set S_i of Type 2 equivalence classes, separate quantified

columns according to the quantifiers over which they range. The separation results in:
EQV1: {{E1.NO}, {E2.NO}} and EQV2: {{E1.SAL}, {E2.SAL}}.

6. Enumerate all combinations of quantified columns KEY1, ..., KEYn over Qi. There should be at most one combination, and there should be at most a single
5 quantified column over Qi in an equivalence class. In the example, there is a single combination for E2 in the example: (E2.NO, E2.SAL).

7. Retain only those Qi where KEYi comprises key columns. In the example, the combination (E2.NO, E2.SAL) forms a key.

8. Compute the derivation of Superset columns from which KEYi is
10 derived and append them to KEYi. In the example, the derivation is {(E2.NO, E2.SAL), ((E5.NO, E5.SAL), (E6.NO, E6.SAL)), ((EMP.NO, EMP.SAL), (RICHEMP.NO, RICHEMP.SAL))}.

9. For each quantifier Qj in the SELECT operator where $Q_j \triangleleft Q_i$, if Qj has a quantified column in each Type 2 equivalence class, then perform the following steps.
15 (Note that there might be more than a single quantified column referencing Qj in an equivalence class. In the example, quantifier E1 qualifies since it is represented in both equivalence classes EQV1 and EQV2. This leads to more alternatives in forming lists of quantified columns and possible matches.)

A. Compute the derivation of Subset columns from which Qi is
20 derived. In the example, the derivation list for quantifier E1 is {(E1.NO, E1.SAL), ((E3.NO, E1.SAL), (E4.NO, E4.SAL)), ((EMP.NO, EMP.SAL), (RICHEMP.NO, RICHEMP.SAL))}.

B. For each derivation in Qi:
i. For all lists of quantified columns in the derivation:
25 I. If the list of quantified columns does not match any list of quantified columns in KEYi, the rule fails. In the example, the lists match on the pairs {((EMP.NO, EMP.SAL), (RICHEMP.NO, RICHEMP.SAL))}.

10. The rule succeeds. In the example, quantifier E2 passes the condition of the rule. Note that the rule also would have been performed had E1 been used as a candidate quantifier instead of E2.

5 Action

1. For each Type 2 equivalence class:

A. Replace all occurrences of quantified columns over Qi with quantified columns in the same Type 2 equivalence class. In the example, all occurrences of E2.NO are replaced with E1.NO, and all occurrences of E2.SAL are replaced with E1.SAL.

10

2. Remove Qi from the query. In the example, quantifier E2 is removed from the query.

LOGIC OF THE OPTIMIZATION TECHNIQUE

15 FIGS. 5A-5D together are a flowchart illustrating the method of optimizing queries in step 202 of FIG. 2 according to the preferred embodiment of the present invention. Specifically, this flowchart further describes the logic of the rule above that implements the preferred embodiment of the present invention for each quantifier Qi in a query.

20 As noted above, the rule is comprised of two parts. Blocks 500-526 represent the condition parts of the rule, and Blocks 528-532 represent the action parts of the rule. The logic of these different blocks are described in more detail below.

25 Referring initially to FIG. 5A, Block 500 is a decision block that represents the XML middleware system 102 determining whether Qi is a FOR EACH quantifier. If not, control exits the flowchart to terminate the logic; otherwise, control transfers to Block 502.

Block 502 is a decision block that represents the XML middleware system 102 determining whether Qi is defined within a SELECT operator that implements an inner

join. If not, control exits the flowchart to terminate the logic; otherwise, control transfers to Block 504.

Block 504 is a decision block that represents the XML middleware system 102 determining whether there are at least two quantifiers Q_i and Q_j in the SELECT operator. If not, control exits the flowchart to terminate the logic; otherwise, control transfers to Block 506.

Block 506 represents the XML middleware system 102 computing equivalence classes among columns in the SELECT operator. These computed equivalence classes are divided into three classes: (a) Type 1 equivalence classes are sets of quantified columns over quantifiers other than Q_i ; (b) Type 2 equivalence classes are sets of quantified columns that reference Q_i and at least one other quantifier Q_j ; and (c) Type 3 equivalence classes are sets of quantified columns that only reference Q_i .

Block 508 represents the XML middleware system 102, for each set S_i of the Type 2 equivalence classes, separating quantified columns according to the quantifiers over which they range.

Block 510 represents the XML middleware system 102 enumerating quantified columns KEY_1, \dots, KEY_n over Q_i . Note that there might be more than a single quantified column over Q_i in an equivalence class. This causes the rule to fail.

Thereafter, control transfers to Block 512 in FIG. 5B via "A".

Referring to FIG. 5B, Block 512 represents the XML middleware system 102 retaining only those KEY_i that comprise key columns.

Block 514 is a decision block that represents the XML middleware system 102 determining if no such retained combination KEY_i exists (i.e., for the condition part of the rule to be true, there must be at least one retained combination KEY_i). If so, control exits the flowchart to terminate the logic; otherwise, control transfers to Block 516.

Block 516 represents the XML middleware system 102 computing the derivation of Superset columns from which KEY_i is derived and adding them to KEY_i .

Block 518 is a decision block that represents the XML middleware system 102

performing a loop for each quantifier Q_j in the SELECT operator where $Q_j \supset Q_i$, if Q_j has a quantified column in each Type 2 equivalence class. For each iteration of the loop, control transfers to Block 520; upon completion of the loop, control transfers to Block 528 in FIG. 5D via "B".

5 Block 520 represents the XML middleware system 102 computing the derivation of Subset columns from which Q_i is derived.

 Thereafter, control transfers to Block 522 in FIG. 5C via "C".

 Referring to FIG. 5C, Block 522 is a decision block that represents the XML middleware system 102 performing a loop for each derivation in Q_i . For each iteration
10 of the loop, control transfers to Block 524; upon completion of the loop, control transfers to Block 518 in FIG. 5B via "C".

 Block 524 is a decision block that represents the XML middleware system 102 performing a loop for all lists of quantified columns in the derivation. For each iteration of the loop, control transfers to Block 526; upon completion of the loop, control
15 transfers to Block 522.

 Block 526 is a decision block that represents the XML middleware system 102 determining whether the list of quantified columns matches any list of quantified columns in KEY_i . If not, control exits the flowchart to terminate the logic; otherwise, control transfers to Block 524.

20 Referring to FIG. 5D, Block 528 is a decision block that represents the XML middleware system 102 performing a loop for each Type 2 equivalence class. For each iteration of the loop, control transfers to Block 530; upon completion of the loop, control transfers to Block 532.

 Block 530 represents the XML middleware system 102 replacing all occurrences
25 of quantified columns over Q_i with quantified columns in the same Type 2 equivalence class.

 Block 532 represents the XML middleware system 102 removing Q_i from the query.

Finally, control exits the flowchart to terminate the logic.

CONCLUSION

This concludes the description of the preferred embodiment of the invention.

- 5 The following describes some alternative embodiments for accomplishing the present invention. For example, any type of computer, such as a mainframe, minicomputer, or personal computer, could be used with the present invention. In addition, any database management system could benefit from the present invention.

- 10 In summary, the present invention discloses a method, apparatus, and article of manufacture for optimizing left outer join queries. The query optimization technique determines whether a query includes a self join that is transitively derived through table expressions having UNION operators. If so, the query is simplified to eliminate the table expressions and to reduce the query to an equivalent query over tables.

- 15 The foregoing description of the preferred embodiment of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description.